Класс, объект и элементы их системы

Рассмотрим реализацию понятия даты с использованием struct для того, чтобы определить представление даты date и множества функций для работы с переменными этого типа:
struct date { int month,int day, int year; }; // дата: месяц, день, год

date today;
void set_date(date, int, int, int);*
void next_date(date);*
void print_date(date);*
// ...

Никакой явной связи между функциями и типом данных нет. Такую связь можно установить, описав функции как члены:
struct date {
int month, day, year;
void set(int, int, int);
void get(int, int*, int*);*
void next();
void print();
};

Функции, описанные таким образом, называются функциями- членами или методами и могут вызываться только для специальной переменной соответствующего типа с использованием стандартного синтаксиса для доступа к членам структуры.

Класс используется для создания объектов. Основная форма имеет вид:

class имя класса
{
закрытые функции –члены(методы) и переменные
public:
открытые функции, функции-члены (методы)и переменные
}

список объектов;//не является обязательным

Закрытые методы и переменные доступны только для других членов этого класса.

Открытые методы и переменные доступны для любой части программы, в которой находится класс.

Функции, объявленные внутри описания класса называются функциями -членами (member functions) или методами.

Для определения методов используется форма:
тип имя класса:: имя метода (параметры)
{
тело функции
}

Два двоеточия после имени класса называются операцией расширения области видимости (scope resolution operator).

Определение класса только определяет тип объектов, а сами объекты не задает) т.е. память для них не выделяется). Для создания объектов имя класса используется как спецификатор типа данных.

После создания объекта к открытым членам класса можно обращаться, используя операцию точка.
Пример.

```
#include <iostream.h>
class class1 {//объвлен класс class1
int a; //доступна для методов class1
public:
int kwadrat(int b);//метод класса class1
};
int class1::kwadrat(int b) //определение метода kwadrat()
{
a=b*b;
return a;
}
main()
{
class1 c; //создается объект с типа class1
cout<<"\n"<<c.kwadrat(3)<<"\n";//вычисление и вывод квадрата трех
return 0;
}
```

Для автоматической инициализации создаваемых объектов в C++ используется функция - конструктор (constructor function), которая включается в описание класса.

Функция конструктор имеет тоже имя, что и класс и не возвращает ни какого значения.

Пример:

```
#include <iostream.h>
// Объявление класса class1
class class1 {
int a;
public:
class1(); // Конструктор
void ft();
};
// Инициализация а конструктором при создании объекта pr
class1::class1()
{
a=100;
}
//Функция возведения в квадрат и печати
void class1::kwadrat()
{
cout << a*a;
}
main()
{
class1 obj//Создание объекта obj
obj.f(); //Вызов функции f()
return 0;
}
```

Как видно из примера конструктор вызывается при создании объекта obj.

Деструкторы. Функция деструктор (destructor)вызывается при удалении объекта для освобождения ресурсов (памяти и т.д.). Она также включается в объявление класса. Перед описанием деструктора ставится значок ~.

Пример.

```
#include <iostream.h>
// Объявление класса class1
class class1 {
int a;
public:
class1(); // Конструктор
~class1(); //Деструктор
void f();
};
// Инициализация а конструктором при создании объекта obj
class1::class1()
{
a=100;
}
//Освобождение ресурсов деструктором
class1::~class1()
{
cout<<"Освобождение\n";
}
//Функция возведения в квадрат и печати
void class1::f()
{
cout << a<<"\n";
}
main()
{
class1 obj;//Создание объекта pr
obj.f(); //Вызов функции f()
return 0;
}
```

Конструкторы с параметрами.

Конструктору можно передать параметры. Для этого нужно добавить необходимые параметры в объявление и определение конструктора. Затем при объявлении объекта параметры задаются в качестве аргумента.

Пример:

```
#include <iostream.h>
class OurClass {
int a;
public:
OurClass(int x); // конструктор
void show();
};
OurClass::OurClass(int x)
{
cout << "В конструкторе\n";
a = x;
}
void OurClass::show()
```

```
{
cout << a << "\n";
}
main()
{
OurClass ob(4);
ob.show();
return 0;
}
```

Конструктор OurClass имеет один параметр. Значение, передаваемое в OurClass() используется для инициализации a. Аргумент 4 передается в ob(4) в качестве аргумента. Деструктор в отличие от конструктора параметров не имеет.

В данном примере конструктору передавали константы, но так же можно передавать переменные:

Пример:

```
#include <iostream.h>
class OurClass {
int i, j;
public:
OurClass(int a, int b);
void show();
};
OurClass::OurClass(int a, int b)
{
i = a;
j = b;
}
void OurClass::show()
{
cout << i << ' ' << j << "\n";
}
main()
{
int x, y;
cout << "Введите два целых: ";
cin >> x >> y;
OurClass ob(x, y);// использование переменных для создания ob
ob.show();
return 0;
}
```

В программе рассмотрено важное свойство объектов. Они могут создаваться по мере необходимости.

```
#include <iostream.h>
class DEMO// Этот класс просто демонстрирует неявные вызовы
// конструктора и деструктора
{
public:
DEMO() { cout « "constructor" « endl; }
~DEMO() { cout « "destructor" « endl; }
};
```

```
void main(void)
{
DEMO staticDemo; // статическое размещение, деструктор вызывается при
// выходе за пределы области видимости
DEMO *dynamicDemo = new DEMO;
// динамическое размещение,
// деструктор вызывается при уничтожении объекта
delete dynamicDemo;
}
```

В этом примере определяется класс, не содержащий ничего, кроме открытых (public) деструктора и конструктора. Обе этих функции объявлены и определены в классе. При создании объекта неявно вызывается конструктор и печатается слово "constructor", а при вызове деструктора, соответственно, слово "destructor".

Внутри функции main() создаются два объекта, один статический, в стеке, а второй в куче (heap) - динамический. В результате выполнения этого примера на экран будет выведено следующее:

```
Constructor
Constructor
Destructor
Destructor
```

Первая строка выводится конструктором при создании объекта staticDemo. Вторая строка выводится конструктором при создании объекта dynamicDemo. Третья строка – результат вызова деструктора объекта dynamicDemo при его уничтожении оператором delete. Четвертая строка – результат вызова деструктора объекта staticDemo. Деструктор статического объекта был вызван последним, при выходе объекта из области видимости.